A Fast and Accurate Non-interactive Privacy-Preserving Neural Network Inference Framework

Hongyao Tao[1], Chungen Xu[1(✉)], and Pan Zhang[2]

[1] School of Mathematics and Statistics, Nanjing University of Science and Technology, Nanjing 210094, China
xuchung@njust.edu.cn

[2] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China

Abstract. With the remarkable successes of machine learning, it is becoming increasingly popular and widespread. Machine learning as a Service (MLaaS) provided by cloud services is widely utilized to address the challenge of users unable to bear the burden of training machine learning models. However, the privacy issues involved present a significant challenge. Homomorphic encryption, known for its capability to perform efficient operations on ciphertexts, is widely employed in Privacy computing domain. In order to address the security vulnerabilities and excessive communication and computation costs of interactive privacy-preserving neural networks, and in light of the significant time consumption of linear layers and the challenges SIMD HE faces in computing arbitrary nonlinear functions precisely, we propose a non-interactive framework for privacy-preserving neural networks that accelerates linear computations and ensures accurate computation of any non-linear functions. Specifically, we utilize CKKS encryption to enable private neural network inference under floating-point arithmetic. Leveraging the characteristics of both wordwise HE and bitwise HE, we design a non-interactive and fast matrix multiplication scheme, achieving up to 500× acceleration across different matrix dimensions. By transforming various types of homomorphic encryption ciphertexts and employing lookup tables, we realize accurate computation of arbitrary non-linear operations without requiring interaction. Experimental results demonstrate that our framework achieves the same level of accuracy as pre-trained neural network models on plaintext without incurring any additional accuracy loss.

Keywords: Homomorphic encryption · Privacy-preserving · Neural networks

Supported by the National Natural Science Foundation of China (No. 62072240) and the Natural Science Foundation of Jiangsu Province Youth Project (No. BK20210330).

J. Liu et al. (Eds.): TridentCom 2023, LNICST 523, pp. 154–172, 2024.
https://doi.org/10.1007/978-3-031-51399-2_9

1 Introduction

As the Internet of Things (IoT) undergoes rapid development, a substantial volume of data is being generated and collected, containing rich information and value. Traditional methods are not capable of effectively handling such massive datasets. Machine learning has emerged as a new technique that can learn patterns and regularities from data. It leverages large-scale data to train models and extract valuable information and knowledge, enabling computers to learn from data and make intelligent decisions. In recent years, machine learning has become increasingly popular and widespread, witnessing an explosive growth in applications. Currently, machine learning have mature applications in various fields, such as image classification [22, 34], medical diagnosis [2], speech recognition [24], and autonomous driving [5]. However, designing and training machine learning models require significant resources and specialized expertise. Extracting value from data through training incurs high resource costs and management expenses. For personal computers or general servers, this could be an overwhelming burden. Fortunately, the rise of new-generation information technologies, such as cloud services, offers a solution. In scenarios with limited local resources, both businesses and individuals can opt to send data to cloud servers and utilize machine learning models pre-trained by cloud server. Machine Learning as a Service (MLaaS) [33] consequently emerged. Recently, with the introduction of AI products like GPT by some internet companies, the momentum of Machine Learning as a Service (MLaaS) has reached new heights.

Meanwhile, while cloud computing brings great convenience to people, data security and privacy have also become increasingly significant concerns in society. The privacy and security issues related to machine learning models and the entire process from training to application in artificial intelligence have attracted widespread attention from researchers. To address the frequent occurrence of data security incidents, many countries have enacted laws and regulations to restrict the sharing of sensitive user data [11]. While clients are obligated to share their data with the cloud for inference, and they expect a high level of data privacy protection to prevent curious cloud providers or attackers from extracting valuable information. Conversely, cloud server aim to prevent users from extracting their model parameter, which have been trained with considerable resources and efforts [30]. Consequently, an imperative exists to devise a secure and efficient solution that safeguards the security of data in Machine Learning as a Service (MLaaS). Considering these challenges, researchers are actively working towards establishing a secure and privacy-preserving framework to facilitate the implementation of MLaaS. To address this issue, various specialized encryption schemes, such as searchable encryption [16], have been applied in diverse scenarios like the Internet of Things [25] and safeguarding privacy data in cloud services [35]. Among these schemes, homomorphic encryption [12] stands out as the most suitable encryption technique for privacy-preserving machine learning (PPML) scenarios that involve computations on encrypted data.

1.1 Related Work

Homomorphic Encryption (HE) is a form of encryption that allows users to perform a series of linear operations directly on encrypted data, with the results still being in

encrypted form. Decryption of the results yields the same content as operating on plaintext data. CryptoNets [14] and its improved version Faster CryptoNets [8] proposed a privacy-preserving scheme based on homomorphic encryption. They utilized Stone-Weierstrass theorem to derive an approximate polynomial for the non-linear activation function to simulate neural network computations on encrypted data, thereby achieving encrypted neural network inference. To ensure accuracy in the non-linear layer, high-order polynomials might be required to approximate the activation functions, resulting in significant overhead for homomorphic encryption. Some approaches [37] that use polynomial approximations update the weights after each iteration and send them to the involved parties for decryption and re-encryption, leading to high communication complexity. CryptoDL [17] improved this by adopting low-order polynomial approximation methods. Kim et al. [20] proposed the first homomorphic encryption logistic regression outsourcing model using low-order polynomials to approximate non-linear function computations. To enhance the efficiency of encrypted computations, E2DM [18] used a more efficient HE scheme, specifically packed SIMD [29], which combines several messages into a singular ciphertext, thereby improving computational efficiency.

However, not all non-linear function can be effectively approximated using polynomials. Therefore, some approaches utilize secure multi-party computation to achieve the computation of non-linear layers with good results. Garbled Circuits are employed by DeepSecure [28] and XONN [27] to binaryize computations in neural networks, allowing them to implicitly derive predictions without revealing sensitive client data. Utilizing Secure Secret Sharing as detailed in [31], client data is partitioned into shares, with the server holding only one share, and the computations are finalized through interactive share exchanges. Due to the nature of secure multi-party computation, the sharing of intermediate results is required. As a black-box oracle for model extraction [32], the server's prediction service is susceptible to exploitation by malicious clients, who may also infer the training dataset [26], resulting in potential harm to the cloud server's model. Substantial computational and communication overhead is incurred during the interaction between the client and the cloud server, which goes against the original intention of outsourcing data to the cloud server due to limited local resources. Consequently, we aim to construct a machine learning as a service (MLaaS) scheme which requires no additional interaction beyond sending encrypted data and receiving prediction results.

In the existing homomorphic encryption schemes, the second-generation homomorphic encryption (word-wise HEs) [4, 6, 10] offers efficient polynomial operations, such as high-speed addition and multiplication, through support for SIMD batch processing. However, the absence of support for non-linear operations, such as the sigmoid and ReLu activation functions in neural networks, commonly used in machine learning processes, is a notable limitation. On the other hand, the third-generation homomorphic encryption scheme (bitwise HEs) [7, 9, 13] supports arbitrary functions represented as Boolean circuits. Nevertheless, due to its encryption process being done bit by bit based on message values, it is impractical for addition and multiplication which will result in extremely low computational efficiency. This can result in unbearable computational costs. To address these limitations, CHIMERA [3] and PEGASUS [23] strike a balance between the advantages and disadvantages of both types of homomorphic encryption schemes. They propose conversion methods between word-wise HEs and bit-wise HEs encryption

schemes, enabling bridging between polynomial and non-polynomial computations in homomorphic encryption.

1.2 Contribution

It is worth noting that existing research points out that in privacy-preserving neural network models, linear computations occupy the vast majority of computation time, accounting for over 90% in many models, and even higher in some cases. Several effective improvement schemes for matrix multiplication have been proposed in [15, 19]. The most recent work [36] reduces the most time-consuming permutation operation in homomorphic encryption linear computations through secret sharing, significantly improving the efficiency of matrix multiplication calculations.

We propose a non-interactive privacy-preserving neural network scheme which enables faster linear computations and accurate evaluation of any non-linear operation. To achieve this, we utilize the CKKS homomorphic encryption scheme for machine learning inference, allowing homomorphic operations on floating-point numbers. Additionally, we design a fast matrix multiplication algorithm based on the characteristics of two types of homomorphic encryption ciphertexts, which achieves theoretically the same or even better results compared to schemes utilizing secret sharing [36], but without requiring any interactions. Our experiments demonstrate that our approach achieves a 500× speedup for matrix multiplication compared to [19]. Furthermore, we leverage table lookup to implement non-linear operations for any activation function by transforming different types of homomorphic encryption ciphertexts. Experimental results show that our proposed scheme achieves the same level of precision as pre-trained neural network models on plaintext, without incurring any additional losses.

Organized as follows, the remaining sections of this paper delve into the study's prerequisites in Sect. 2, our approach's specific details in Sect. 3, the corresponding experimental results in Sect. 4, and a conclusion in Sect. 5.

2 Preliminaries

2.1 Basic Notations

Vectors are denoted using bold lowercase letters, such as $\mathbf{a}$, and $\mathbf{a}_j$ is used to denote the j-th component of vector a. Matrices are denoted using bold uppercase letters, such as $\mathbf{A}$, where $\mathrm{a}_{i,j}$ signifies the (i, j) entry of matrix $\mathbf{A}$. The Hadamard product of vectors is represented as $\mathbf{a} \circ \mathbf{b}$. We represent the set $\{0,\ldots, q-1\}$ as $\mathbb{Z}_q$, and $\mathbb{Z}_q[X]$ denotes the polynomial ring over $\mathbb{Z}_q$. In the case of a 2-power number n, $R_{n,q} \equiv \mathbb{Z}_q[X]/(X^n+1)$ specifies the ring of polynomials with a degree less than n, with coefficients belonging to $\mathbb{Z}_q$. Lower-case letters, like a, are employed to symbolize elements in $R_{n,q}$, with a_j denoting the j-th coefficient of a. The multiplication of ring elements is indicated by the dot symbol $\cdot$, as in $a \cdot b$. In this paper, all logarithms are to base 2.

2.2 Homomorphic Encryption

Homomorphic encryption possesses a unique property in ciphertext processing, which allows for efficient operations on encrypted data without decryption. This feature is pivotal in upholding information security and introduces a novel approach to tackle the data security challenges encountered in cloud computing. Next we will introduce two commonly used schemes for homomorphic encryption, namely LWE encryption (bit-wise HEs) and RLWE encryption (word-wise HEs).

The notation $\mathrm{LWE}^{n,q}{}_{\mathrm{s}}(m)$ is employed to represent the LWE encryption of the message $m \in \mathbb{Z}_q$ using the secret key $\mathrm{s} \in \mathbb{Z}_q{}^n$, and abbreviated as $[m]_1$. In the same vein, the RLWE encryption of the message $m \in R_{n,q}$ under the key $s \in R_n$ is denoted as $\mathrm{RLWE}^{n,q}{}_s(m)$, with a shorthand notation of $[m]_2$. It is important to note that all LWE and RLWE ciphertexts belong to the ring R_n, q. Next we introduce some linear operations for homomorphic encryption.

- Addition (+) and Multiplication ($\cdot$). In (R)LWE, if we have ciphertexts c_0 and c_1, encrypting ring elements r_0 and r_1, the operation $\mathrm{c}_0 + \mathrm{c}_1$ (or $\mathrm{c}_0 \cdot \mathrm{c}_1$) yields a ciphertext encrypting $r_0 + r_1$ (or $r_0 \cdot r_1$).
- PtMult($\diamond$). Subject to the operation $r \diamond \mathrm{c}_0$, the ciphertext c_0, encrypting a ring element r, transforms into a ciphertext that encrypts $r \cdot t$, where t is a given plaintext element.

In the case of RLWE encryption schemes, such as CKKS, it is also possible to combine several messages into a singular ciphertext and efficiently perform homomorphic computations using Single Instruction Multiple Data (SIMD) techniques. Unlike other homomorphic encryption schemes where the plaintext domain is integers, CKKS is an RLWE-based homomorphic encryption scheme that allows packed encryption computations on a series of floating-point numbers. This characteristic makes CKKS particularly suitable for privacy-preserving solutions during the computing process.

Leveraging the Chinese Remainder Theorem in the ring $R_{n,q}$, we can represent multiple plaintext data as a vector. This vector is subsequently encoded into a polynomial on the ring $R_{n,q}$, enabling various encryption operations on this polynomial. This methodology enables the completion of the SIMD batch processing described above. Here are some commonly used operations based on SIMD technology:

- SIMD Addition and Multiplication. In the addition $\mathrm{c}_0 + \mathrm{c}_1$ (or multiplication $\mathrm{c}_0 \cdot \mathrm{c}_1$) operation, RLWE ciphertexts c_0 and c_1 encrypting vectors $\mathbf{u}$ and $\mathbf{v}$ result in a ciphertext c. This ciphertext encrypts the vector $\mathbf{u} + \mathbf{v}$ (or $\mathbf{u} \circ \mathbf{v}$).
- Rotation. In the presence of an RLWE ciphertext c encrypting the vector $\mathbf{v}$, an integer $n \in \mathrm{N}$, and an evaluation key, the operation $Rot(\mathbf{c}, n)$ generates an RLWE ciphertext. This ciphertext encrypts the vector obtained by left-shifting all components of $\mathbf{v}$ by n positions concurrently.
- Switch. The RLWE ciphertext c encrypting the vector $\mathbf{v}$ undergoes the Swt operation, leading to a new RLWE ciphertext. This ciphertext encrypts a ring element v with coefficients $v_i = \mathbf{v}_i$ across all possible positions.
- Transform. The operation $Transf\ (c, i)$, applied to the RLWE ciphertext $c \in \mathrm{RLWE}\ \mathrm{s}^{n,p}(m)$ and an integer $i \in n$, results in the transformation of c into an LWE-encrypted ciphertext $\mathrm{LWE}\ \mathrm{s}^{n,p}\ (m_k)$. This LWE ciphertext corresponds to the i-th coefficient of m under the same key.

2.3 System Model

In the MLaaS (Machine Learning as a Service) system illustrated in Fig. 1, private data is in the possession of the client. The cloud server, featuring a trained deep learning model, provides inference services using data received from the client. For instance, a doctor transmits encrypted patient data to the server. The server runs a neural network model and sends encrypted predictions back to the doctor. After decryption, it can help doctor to analyse the diagnostic process and the creation of medical rehabilitation plans.

We focus lies on neural networks, which have achieved widespread success and demonstrated remarkable performance in image classification and face recognition. A neural network is composed of layers that learn the complex relationships within input data. It operates on a series of linear and non-linear transformations to infer results. For instance, given a patient's medical data as input, it can determine whether the patient has a particular disease. Linear transformations come in two forms: matrix multiplication and convolutions. Non-linear transformations employ activations to approximate complex functions and pooling to reduce dimensions. Neural networks iteratively apply linear and non-linear transformations, reducing the dimension of input data, and ultimately producing classification results.

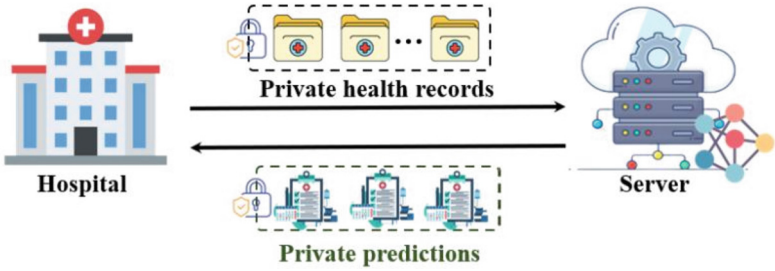

Fig. 1. A schematic diagram of an MLaaS system.

The discussion that follows will specifically target image classification, employing it as an illustration to convey a distinct comprehension of neural network architecture, visually depicted in Fig. 2.

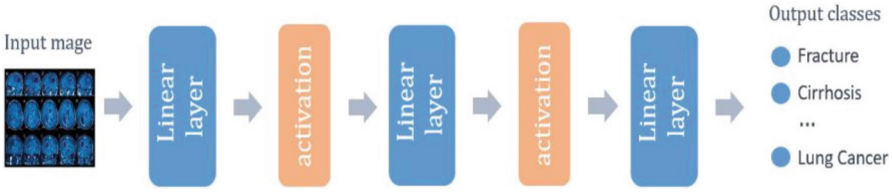

Fig. 2. A schematic diagram of an neural network.

Matrix multiplication is indeed the most common form of linear layer in neural networks. In image classification tasks, the input data is typically represented as a matrix

or a tensor, where each element corresponds to a pixel or a feature. The linear layer in the network performs a matrix multiplication operation between the weight matrix and input data, followed by the addition of a bias term. Executing matrix multiplication involves taking the dot product of the weight matrix W with dimensions $s \times l$ and a input vector data of size $l \times 1$. This operation produces a vector with dimensions $s \times 1$.

Non-linear activation functions are commonly applied after the linear layer to introduce non-linearity and enable the network to capture more complex relationships in the data. Activation functions frequently used include ReLu, given by $f(x) = \max 0, x$; sigmoid, denoted as $f(x) = \frac{1}{1+e^{-x}}$; and tanh, expressed as $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$. This combination of the linear and non-linear operations is the fundamental building block in neural network architectures for image classification and other tasks.

2.4 Threat Model

We have outlined the requisite security properties, adopting a semi-honest model akin to previous works [19, 36]. In this model, both parties seek to extract additional information from the received messages, assuming limited computational capabilities. In simpler terms, while following the protocol, the client endeavors to discover model parameters, while the cloud server attempts to gain insights into the client's data.

3 System Description

3.1 Linear Layers

The linear layer in a neural network plays a crucial role by combining the input data with weights through matrix multiplication and adding biases to learn a new representation of the data. Homomorphic encryption-based linear computations, comprising a sequence of Add, Mult, and Perm operations, have consistently emerged as the predominant component in privacy-preserving neural network models, as indicated by research experiments. In this context, encrypted vectors from the client and plaintext weight matrices from the server serve as inputs, resulting in an encrypted dot product as the output. Among these, Perm operations is the most time-consuming part of linear operations, making it essential to reduce permutation operations to improve the efficiency of neural networks. We have designed a non-interactive approach to reduce the transpose operation in matrix multiplication during linear operations. Moreover, this approach can be directly applied to other homomorphic encryption-based linear computations, such as convolution operations, and yields significant efficiency improvements.

We start with the most straightforward explanation of the matrix-vector multiplication, which is slow and inefficient. Then, we describe optimizations to make the operation faster. To facilitate comparison, we adopt a system framework commonly used in previous approaches. Specifically, we consider a matrix with dimensions s rows and l columns, representing a linear layer with s input elements and l output elements. The number of slots for ciphertext is denoted by n. For ease of discussion, we assume that n, s, and l are powers of 2 (otherwise, we can pad them with zeros to the nearest power of 2). Typically, to enhance computational efficiency, we set n to be sufficiently large for

batch processing of more data. Thus, without loss of generality, we reasonably assume that both s and l are smaller than n.

The Naive Method: We consider the matrix-vector multiplication process of $\mathbf{A}\mathrm{x} = \mathbf{b}$, as depicted in Fig. 3. Here, $\mathbf{A}$ is an $s \times l$ plaintext matrix, representing the neural network model weights held by the server; the client offers an RLWE-encrypted vector $[\mathbf{x}]_\mathbf{1}$ as a representation of the uploaded data. After the matrix multiplication, the result is a homomorphically encrypted vector $[\mathbf{b}]_\mathbf{1}$. In the naive model, individual plaintext vectors $\mathrm{a_i}$ are created by the server through encoding each row of matrix $\mathbf{A}$ separately. Then the server performs homomorphic multiplication with the encrypted vector$[\mathbf{x}]_1$ for each plaintext vector $\mathbf{a_i}$, yielding intermediate results $[\mathbf{v_i}]_\mathbf{1} = [\mathbf{a_i} \circ \mathbf{x}]_\mathbf{1}$. Note that due to the characteristics of homomorphic encryption, the size of all encryption vectors is the number of ciphertext slots n. By summing up each element of $\mathbf{v}_\mathrm{i}$, we obtain the j-th element $\mathbf{b}_j$ of the corresponding result vector $\mathbf{b}$.

To derive the final result vector b from matrix multiplication, we must individually sum up the components of each intermediate vector $\mathbf{v_i}$. However, due to the characteristics of SIMD technology, encoding the vectors in a packed manner results in a corresponding ring element. Performing the necessary operations on the ring element to obtain the sum is not straightforward. Therefore, we rely on the permutation operation described earlier. As depicted in Fig. 3, during an operation named Ras, we cyclically rotate the positions of the intermediate vector $\mathbf{v_i}$ and then conduct SIMD addition. By repeating this process multiple times, we eventually obtain a ciphertext. Each component of the encrypted vector corresponds to the j-th component $\mathbf{b}_j$ of the result vector. Thus, a total of s Ras operations are needed, generating s ciphertexts, each corresponding to one of the s components of the encrypted result vector. In the meantime, if we want to consolidate these s ciphertexts into one, we can achieve this by taking the sum of the result obtained from multiplying each of the s ciphertexts with a unit vector (a vector with only one element as 1 and all other elements as 0). The resulting ciphertext $[\mathbf{b}]_\mathbf{1}$ will be the final encrypted result.

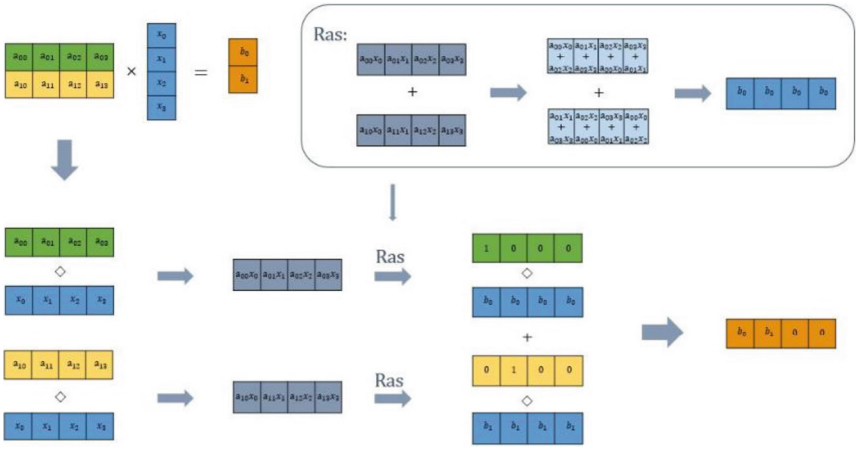

Fig. 3. A schematic diagram of the Naive Method.

In this process, the server is able to compute the dot product between the plaintext weight matrix **A** and the encrypted input vector $[\mathbf{x}]_1$, and obtain the encrypted dot product $[\mathbf{b}]_1$ without learning the plaintext data $[\mathbf{x}]_1$. This ensures the privacy protection of the input data vector during computation. In the naive model process, without considering the final multiplication with the unit vector, there are a total of s SIMD PtMult operations, $s \log l$ Permutation operations and $s \log l$ SIMD Addition operations.

The Diagonal and Hybrid Method: The Diagonal Model [15] offers another potential solution to the problem of multiple output ciphertexts. The fundamental concept of this method involves representing the elements of the plaintext weight matrix **A** as s vectors ($\mathbf{a_0}$, $\mathbf{a_1}$,…, $\mathbf{a_{s-1}}$) in accordance with the diagonal order. Then, each vector $\mathbf{a_i}$ is multiplied with a different transposed ciphertext vector $[\mathbf{x_i}]_1$, and finally, all corresponding intermediate result cipher-text vectors $[\mathbf{v_i}]_1$ are summed to obtain the corresponding final result vector **b**. This Diagonal Model efficiently consolidates the results from multiple output ciphertexts into a single ciphertext $[\mathbf{b}]_1$. And the Diagonal Method does not require Ras operation, reducing the complexity and providing a possible solution for handling multiple encrypted output results in privacy-preserving computations based on homomorphic encryption.

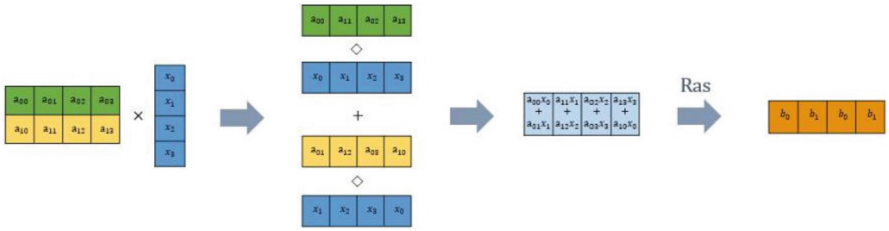

Fig. 4. A schematic diagram of the Hybrid Method.

In order to improve the efficiency of homomorphic encryption linear computation and reduce the waste of resources in empty ciphertext slots, GAZELLE [19] proposes a hybrid approach that combines Diagonal Encoding with RaS. In this scheme, the server first encodes matrix **A** into n_0 plaintext vectors using the diagonal encoding method. The first plaintext vector $\mathbf{a_0}$, as depicted in Fig. 4, is formed by the diagonal elements ($\mathbf{a_{0,0}}$, $\mathbf{a_{1,1}}$, $\mathbf{a_{0,2}}$, $\mathbf{a_{1,3}}$) extracted from the matrix **A**. The second plaintext vector a_1 is comprised of the remaining diagonal elements ($\mathbf{a_{0,1}}$, $\mathbf{a_{1,2}}$, $\mathbf{a_{0,3}}$, $\mathbf{a_{1,0}}$).

After preprocessing the plaintext weight data with the diagonalization packing as described above, we employ SIMD technology to bundle it into a ring element for subsequent SIMD PtMult operations with the input ciphertext vector. As depicted in Fig. 4, it is evident that we no longer require Ras operations for each intermediate vector after the SIMD PtMult operation. By performing SIMD addition on these intermediate vectors, a single Ras operation yields the final result vector. This approach reduces the number of Ras processes from s to 1, significantly enhancing the efficiency of linear computations under ciphertext. Moreover, unlike the original model that produced s ciphertext results, this method directly obtains the final ciphertext result vector b from the matrix multiplication.

As explained in the preceding sections, to boost batch processing efficiency, the number of ciphertext slots n is typically set to a larger value. To make full use of the available ciphertext slots, GAZELLE [19] enhances the efficiency of linear computations by packing multiple input vectors into one. Specifically, n/l input vectors can be packed into one input vector, and correspondingly, n/l weight vectors can be packed into a diagonal encoding matrix for subsequent matrix multiplication operations. This process transforms into a preprocessed matrix with $s \times l/n$ rows and n columns, multiplied by a packed $n \times 1$ input encrypted vector. Similarly, by performing a Ras operation on the vector after SIMD addition, the final ciphertext result vector b can be obtained directly. The hybrid method requires a total of $s \times l/n$ SIMD PtMult operations, $s \times l/n - 1 + \log n/s$ Permutation operations, and $s \times l/n - 1 + \log_{n/s}$ SIMD addition operations.

Our Fast Method: In the subsequent GALA [36] model, the authors proposed utilizing a method based on secret sharing to enable the RaS operation to be performed in plaintext, thereby significantly reducing the transposition operations in matrix multiplication. [19] and [36] categorized the permutation in matrix multiplication into HstPerm and Perm. It is generally preferred to have a larger value of n to pack more data efficiently and achieve high-performance SIMD HE. However, in the hybrid method, with the increase in the number of ciphertext slots n, the required number of permutation operations proportionally escalates, significantly augmenting the computational overhead in ciphertext-based calculations. Therefore, GALA introduces a novel row-wise weighted matrix encoding scheme, combined with secret sharing techniques, which dramatically reduces the number of permutation operations in ciphertext-based linear computations, thereby enhancing computational efficiency. In GALA's scheme, the cost of transpositions in matrix multiplication is reduced to only $s \times l/n - 1$ Permutation rotations, significantly decreasing the overhead required for linear computations.

We propose a novel scheme for fast linear computations. Similar to GALA's approach, which leverages secure multi-party computation to eliminate permutations in RaS, we discovered that using lookup tables can achieve the same effect. Additionally, in our scheme, the cost of permutations in matrix multiplication is reduced to only $s \times l/n - 1$ HstPerm rotations rather than $s \times l/n - 1$ Perm rotations, which is lower than the cost required by GALA. Figure 5 demonstrates how our proposed scheme performs matrix-vector computations, emphasizing its efficiency in managing permutations and enhancing linear calculations.

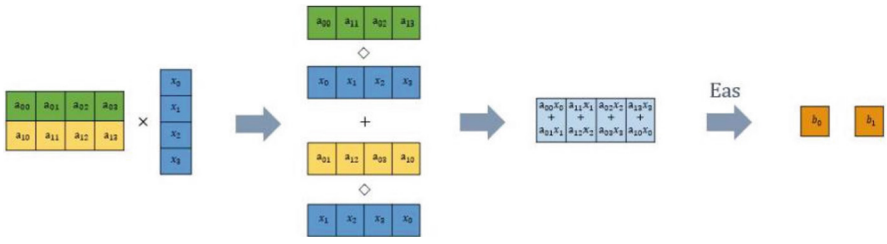

Fig. 5. A schematic diagram of our Fast Method.

Same as GAZELLE, the server first encodes matrix $\mathbf{A}$ into n_0 plaintext vectors using the diagonal encoding method. The first plaintext vector a_0 is formed by the diagonal elements ($\mathbf{a_{0,0}}$, $\mathbf{a_{1,1}}$, $\mathbf{a_{0,2}}$, $\mathbf{a_{1,3}}$) extracted from the matrix A. The second plaintext vector a_1 is comprised of the remaining diagonal elements ($\mathbf{a_{0,1}}$, $\mathbf{a_{1,2}}$, $\mathbf{a_{0,3}}$, $\mathbf{a_{1,0}}$). By performing a series of permutation operations on the encryption vector $[\mathbf{x}]_\mathbf{1}$, and uses SIMD PtMult to perform elementwise multiplication with $\mathbf{a}_i$. As a result, the server gets s intermediate ciphertext vector, $[\mathbf{v_i}]_\mathbf{1} = [\mathbf{a_i}\ \mathbf{x^1}]_\mathbf{1}$.

Next, what differs is that after performing the SIMD Addition operation on $[\mathbf{v_0}]_\mathbf{1}$ and $[\mathbf{v_1}]_\mathbf{1}$ to get a new intermediate ciphertext vector $[\mathbf{u}]_\mathbf{1}$, we use the Switch operation to transform the encrypted ciphertext of vector $[\mathbf{u}]_\mathbf{1}$ into an encrypted ciphertext of a ring element u. In this process, each coefficient u_i of the ring element u is represented by $\mathbf{u}_i$. Then, correspondingly, we perform another Transform operation to obtain n LWE-encrypted ciphertexts, each representing the coefficient u_i of the ring element u. Based on this scenario, we can naturally perform homomorphic addition operations on each component of vector u to obtain the LWE-encrypted results of each component of the final matrix-vector multiplication result vector $\mathbf{b}$, i.e., $\mathbf{b}_0$ and $\mathbf{b}_1$. It is worth nothing that the Switch operation and Transform operation are essential for subsequent lookup table operations. Therefore, our approach only introduces the time required for LWE homomorphic addition operation but offsets the SIMD PtMult operations and SIMD Addition operations required by RaS. Moreover, LWE homomorphic addition operation can be embedded into the lookup table process. As the lookup table undergoes inherent key conversion operations before performing LUT operations, it significantly reduces the dimension of the ciphertext. Our experiments show that the time required for one LWE homomorphic addition operation is only a tiny fraction of the SIMD Addition operations under the original CKKS ciphertext dimension, greatly reducing the overhead of linear computations. Similarly, we enhance the utilization of ciphertext slots by packing multiple input vectors x into a single ciphertext and multiple different rows into a single plaintext vector. Consequently, our approach requires $s \times l/n$ SIMD PtMult operations, $s \times l/(n-1)$ SIMD Addition opetations, $s \times l/(n-1)$ Permutation rotations, and $s(l-1)$ LWE additions (Add). The result is s LWE ciphertexts, which are utilized for the subsequent non-linear layer's lookup table operations. Table 1 compares the computational overhead of different models. We can see that our fast method dramatically reduces the computational complexity.

Table 1. Computational overhead of different models.

Method	Perm	Mult	Add	LWE Add
Naive	$s \log l$	s	$s \log l$	0
GAZELLE [19]	$\frac{sl}{n} - 1 + \log \frac{n}{s}$	$\frac{sl}{n}$	$\frac{sl}{n} - 1 + \log \frac{n}{s}$	0
Our	$\frac{sl}{n} - 1$	$\frac{sl}{n}$	$\frac{sl}{n} - 1$	$s(l-1)$

3.2 Non-linear Layers

The non-linear layers in a neural network introduce non-linear transformations, enabling the network to learn complex non-linear relationships. The incorporation of these non-linear activation functions allows neural networks to express more intricate non-linear relationships, thereby enhancing the network's representational capacity and enabling it to better handle various complex machine learning tasks. In order to avoid the significant communication and computational overhead of interacting non-linear layers in secure multi-party computation schemes, we aim to perform neural network inference using fully homomorphic encryption throughout the process.

Meanwhile, to strike a balance between word-wise HEs that efficiently perform SIMD homomorphic computations by packing multiple plaintexts into one ciphertext and bit-wise HEs that support arbitrary functions represented by boolean circuits, we adopt the approach from [23] to implement the conversion between word-wise HEs ciphertexts and bit-wise HEs ciphertexts. Furthermore, we utilize the lookup table based on LWE ciphertexts to perform arbitrary non-linear computations. We embed the fast matrix addition operation discussed in the previous section before the lookup table operation, as illustrated in Algorithm 1.

Algorithm 1: ReLu activation functions

Data: $\mathrm{RLWE}_s^{n,q}(\mathbf{u})$, look-up table function $\mathrm{ReLu}(x)$
Result: $\mathrm{RLWE}_s^{n,q}(\mathrm{ReLu}(\mathbf{b}))$

Switch: $ct' = \mathrm{RLWE}_s^{n,q}(u)$
for $i \leftarrow 1$ **to** n
$ct_i = Transf^i(\mathrm{RLWE}_s^{n,q}(u) = \mathrm{LWE}_{\mathbf{s}}^{n,q}(u_i)$
Key-Switch: $ct_i = KS(\mathrm{LWE}_{\mathbf{s}}^{n,q}(u_i)) = \mathrm{LWE}_{\mathbf{s}'}^{n',q}(u_i)$
end for
homomorphic addition: $ct_j = \mathrm{LWE}_{\mathbf{s}'}^{n',q}(\mathbf{b}_i)$ for all $j \in \langle n_0 \rangle$
for $j \leftarrow 1$ **to** n_0
Evaluate the look-up table in parallel:
$ct'_j = LUT(\mathrm{LWE}_{\mathbf{s}'}^{n',q}(\mathbf{b}_i)) = \mathrm{LWE}_{\mathbf{s}'}^{n',q}(\mathrm{ReLu}(\mathbf{b}_i))$
end for
Repacking: $ct = Repack(\mathrm{LWE}_{\mathbf{s}'}^{n',q}(\mathrm{ReLu}(\mathbf{b}_i))) = \mathrm{RLWE}_s^{n,q'}(\mathrm{ReLu}(\mathbf{b}))$

We use the ReLu activation function as an example to illustrate our non-linear layer process. Firstly, we use the Switch operation to transform the encrypted ciphertext of vector $[\mathbf{u}]_{\mathbf{C}}$ into an encrypted ciphertext of a ring element u. In this process, each coefficient u_i of the ring element u is represented by u_i. Then, correspondingly, we perform another Transform operation to obtain n LWE-encrypted ciphertexts, each representing the coefficient u_i of the ring element u. Simultaneously, we reduce the dimensionality of the ciphertexts through key switching operations to enhance the efficiency of subsequent operations. Based on this scenario, we can naturally perform homomorphic addition operations on each component of vector $\mathbf{u}$ to obtain the LWE-encrypted results of each component of the final matrix-vector multiplication result vector $\mathbf{b}$, i.e., $\mathbf{b_0}$,…, $\mathbf{b_1}$. Next, we perform the lookup table operation with the ReLu activation function

on n_0 ciphertexts $\mathrm{LWE}_{s'}^{n',q}(\mathbf{b}_i)$, obtaining n_0 corresponding result ciphertexts $\mathrm{LWE}_{s'}^{n',q}(\mathrm{ReLu}(\mathbf{b}_i))$. Finally, we utilize a repackaging function to bundle the n_0 result ciphertexts into a single RLWE ciphertext $\mathrm{RLWE}_{s'}^{n,q'}(\mathrm{ReLu}(\mathbf{b}))$ representing an encrypted vector $\mathbf{b}$.

3.3 Noise Management

Homomorphic encryption ensures the security of plaintext messages by introducing noise. Performing a series of homomorphic operations on ciphertexts will lead to an increase in noise. In the event that the noise goes above a designated threshold, there is a risk of decryption failure or errors. Therefore, managing the noise of ciphertexts during computations becomes particularly crucial.

We assume that the initial noise of two ciphertexts are e_0 and e_1, and the noise increases to $e_0 + e_1$ after a SIMD Addition operation. After a single SIMD PtMult operation, the noise increases by a constant factor to $e_{mult}\eta_0$. Similarly, following a permutation rotation operation, the noise rises to ee_0. Typically, we assume $e > e_{mult} \gg e_0$. Table 2 describes the growth of noise for the three schemes presented in Sect. 3.1. In our work, we utilize the hierarchical HE scheme CKKS, ensuring the correctness of ciphertexts after respective computation operations by predefining the required multiplication depth.

Table 2. Noise growth for different models.

Method	Noise growth
Naïve	$le_0e_{mult} + (l-1)e$
GAZELLE [19]	$le_0e_{mult} + \left(l - \frac{n}{s}\right)e_{mult}e + \left(\frac{n}{s} - 1\right)e$
Our	$le_0e_{mult} + \left(l - \frac{n}{s}\right)e_{mult}e$

3.4 Security Analysis

In our scheme, the cloud server can only has the encrypted input data, which prevents data leakage from reconstruction attacks or membership inference attacks, ensuring the protection of client privacy. In the meantime, the client can only obtain the results after homomorphic computations, having no knowledge of the server's computation process. It prevents the exploitation of the server's prediction service as a black-box oracle for extracting model parameters and deducing the training dataset. The security of both the input data and the model parameters relies entirely on the security of HE schemes like CKKS.

Li et al. [21] emphasized that the approximate decryption outcome in CKKS might reveal supplementary details about the decryption key. They effectively devised a passive attack capable of retrieving the decryption key when provided access to the decryption outcomes. Consequently, the client must refrain from divulging the decryption results or distributing them to individuals who aren't entirely trustworthy.

4 Experiments

Table 3. Computational overhead of different operations.

Operation	Perm	Mult	Add	LWE Add
Time(ms)	335.31	5.45	5.05	0.0025

Table 4. Computational overhead of different models.

Method	Perm	Mult	Add	LWE Add	Time (ms)
		Dimension:1 × 1024			
Naive	10	1	10	0	3352.62
[19]	10	1	10	0	3356.32
Our	0	1	0	1023	6.67
		Dimension:8 × 256			
Naive	64	8	64	0	21630.6
[19]	8	2	8	0	2708.51
Our	1	2	1	2040	346.99
		Dimension:16 × 128			
Naive	112	16	112	0	37864.2
[19]	7	2	7	0	2370.61
Our	1	2	1	2032	346.24
		Dimension:16 × 64			
Naive	64	16	64	0	32460.1
[19]	6	1	6	0	2026.82
Our	0	1	0	1008	6.67

We conducted our experiments on Ubuntu 18.04, and have an Intel i7-10700F CPU, running at 2.90 GHz with 16 GB of system memory. The dimension n of the RLWE is 2^{16}, while the dimension n' of the LWE ciphertext after key switching is 2^{10}. The dimension $\underline{n}$ of the ciphertext in the LUT function is also 2^{10}. According to [23] and [1], all selected parameters satisfy at least 119 bits of security. Based on the aforementioned parameter settings, we tested the running time of our scheme and the previous scheme for matrix multiplication under CKKS encryption. Table 3 compares the computational overhead of different operations. We can observe that the time taken for the Perm operation is 61 times that of the Mult operation and 66 times that of the Add operation. Furthermore, the time for Slot-wise Add operation is 2000 times greater than that of LWE Add operation. As shown in Table 4, our approach requires lower computational complexity compared

to other methods, significantly reducing the time required for linear operations. Across various matrix dimensions, the acceleration achieved can reach up to 500×.

Table 5. Model accuracy in different dimensions.

Dimension (n)	10	11	12	Plaintext
Accuracy	97.85%	98.25%	98.25%	98.25%

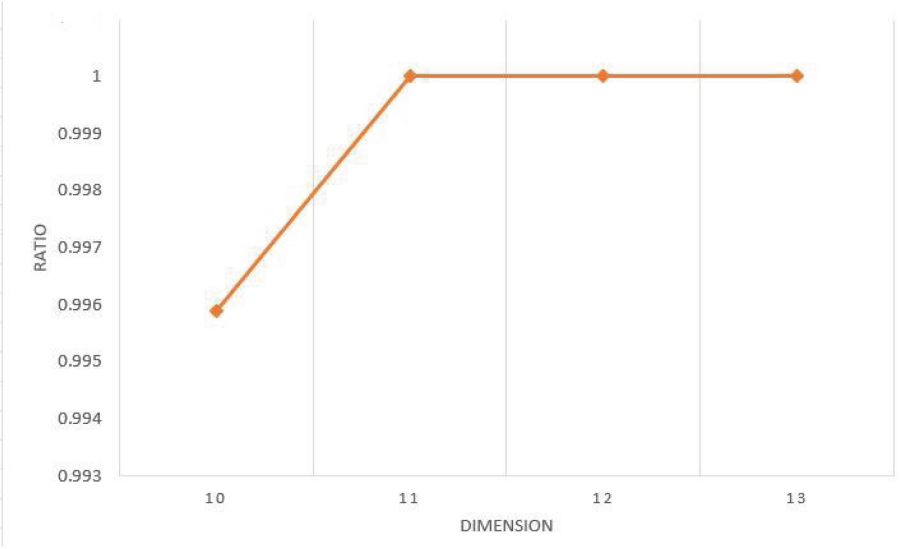

Fig. 6. The ratio of accuracy between inference under ciphertext and plaintext for various dimensions.

We employed PyTorch to pretrain a neural network consisting of three hidden layers on the minst dataset and utilized the model's parameters for making predictions on encrypted data. Table 5 and Fig. 6 illustrate the precision of our proposed model. As the dimension of the LUT increases, the accuracy of our approach improves progressively, achieving results comparable to inference predictions made under plaintext conditions. For other complex models and datasets, increasing the dimension of the LUT can achieve high-precision prediction of the model, at the expense of increased LUT time.

As illustrated in Fig. 7, although the proportion of time taken by the non-linear layer in the model increases with the augmentation of LUT dimension, we can effectively accelerate LUT calculations through parallel computing. This approach significantly reduces the time portion occupied by the non-linear layer in the model. Figure 8 showcases the time distribution of the non-linear layer in the model across different threads in a higher-dimensional ($\underline{n} = 12$) LUT model. It can be observed that with the increase in threads, the time portion of the non-linear layer in the model decreases from 50.57% to 29.81%, resulting in substantial reduction.

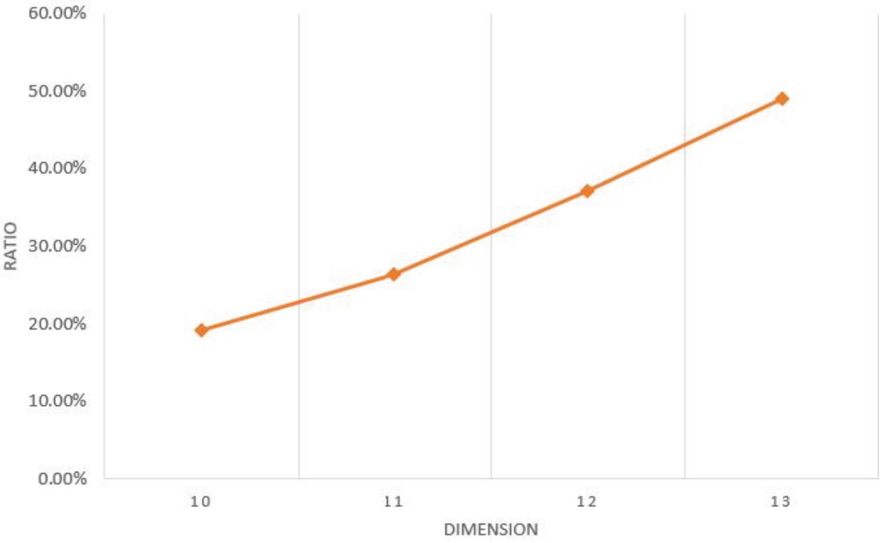

Fig. 7. The proportion of time taken by the non-linear layers across different dimensions.

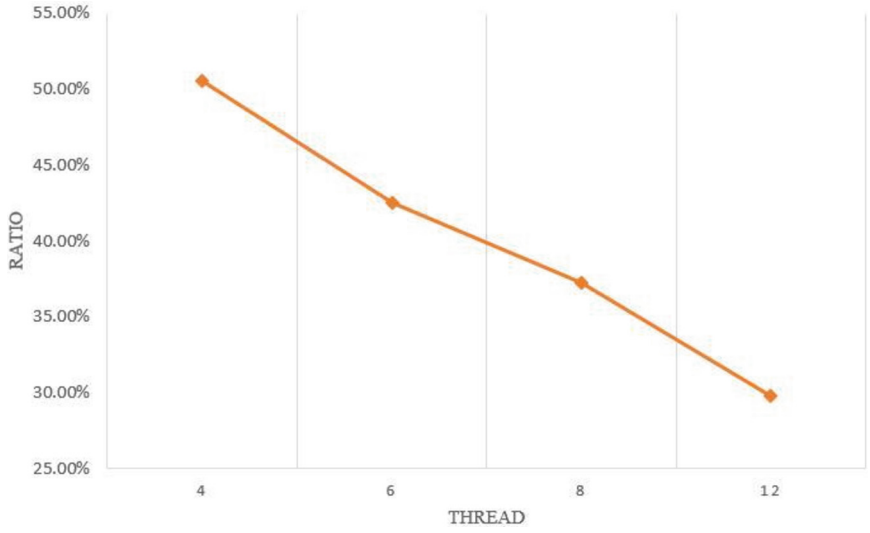

Fig. 8. The proportion of time taken by the non-linear layers under different threads.

5 Conclusion

In this paper, we introduce a non-interactive privacy-preserving neural network framework based on homomorphic encryption. This framework safeguards the security of training data and model parameters for MLaaS in cloud environments, enabling computation and inference over floating-point numbers. Leveraging the characteristics of two

types of homomorphic encryption ciphertexts, we devise a non-interactive and efficient matrix multiplication scheme that achieves up to 500× acceleration across varying matrix dimensions. This scheme significantly reduces the time consumed by the linear layers, which constitute a major portion of time in privacy-preserving neural network models. Additionally, by transforming different types of homomorphic encryption ciphertexts and employing lookup tables, we ensure precise computation of arbitrary non-linear operations without the need for interaction. Consequently, our approach attains the same level of accuracy as pre-trained neural network models on plaintext while incurring no additional accuracy loss.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
2. Arabasadi, Z., Alizadehsani, R., Roshanzamir, M., Moosaei, H., Yarifard, A.A.: Computer aided decision making for heart disease detection using hybrid neural network-genetic algorithm. Comput. Methods Prog. Biomed. **141**, 19–26 (2017)
3. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. J. Math. Cryptol. **14**(1), 316–338 (2020)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (2014)
5. Candela, E., Doustaly, O., Parada, L., Feng, F., Demiris, Y., Angeloudis, P.: Risk-aware controller for autonomous vehicles using model-based collision prediction and reinforcement learning. Artif. Intell. **320**, 103923 (2023)
6. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
8. Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., Fei-Fei, L.: Faster cryptonets: leveraging sparsity for real-world encrypted inference. arXiv preprint arXiv:1811.09953 (2018)
9. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24
10. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch, p. 144 (2012)
11. File, I.: Proposal for a regulation of the European parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (general data protection regulation). General Data Protection Regulation (2012)
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178. ACM (2009)
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5

14. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: ICML. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016)
15. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_31
16. Han, M., Xu, P., Xu, L., Xu, C.: TCA-PEKS: trusted certificateless authentication public-key encryption with keyword search scheme in cloud storage. Peer Peer Netw. Appl. **16**(1), 156–169 (2023)
17. Hesamifard, E., Takabi, H., Ghasemi, M.: Deep neural networks classification over encrypted data. In: CODASPY, pp. 97–108. ACM (2019)
18. Jiang, X., Kim, M., Lauter, K.E., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: CCS, pp. 1209–1222. ACM (2018)
19. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.P.: GAZELLE: a low latency framework for secure neural network inference. In: USENIX Security Symposium, pp. 1651–1669. USENIX Association (2018)
20. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X., et al.: Secure logistic regression based on homomorphic encryption: design and evaluation. JMIR Med. Inform. **6**(2), e8805 (2018)
21. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 648–677. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_23
22. Liu, B., Jing, L., Li, J., Yu, J., Gittens, A., Mahoney, M.W.: Group collaborative representation for image set classification. Int. J. Comput. Vis. **127**(2), 181–206 (2019)
23. Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: IEEE Symposium on Security and Privacy, pp. 1057–1073. IEEE (2021)
24. May, A., et al.: Kernel approximation methods for speech recognition. J. Mach. Learn. Res. **20**, 59:1–59:36 (2019)
25. Mei, L., Xu, C., Xu, L., Yu, X., Zuo, C.: Verifiable identity-based encryption with keyword search for iot from lattice. Comput. Mater. Contin **68**, 2299–2314 (2021)
26. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: IEEE Symposium on Security and Privacy, pp. 739–753. IEEE (2019)
27. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E., Koushanfar, F.: XONN: xnor-based oblivious deep neural network inference. In: USENIX Security Symposium, pp. 1501–1518. USENIX Association (2019)
28. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: scalable provably-secure deep learning. In: DAC, pp. 2:1–2:6. ACM (2018)
29. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptogr. **71**(1), 57–81 (2014)
30. Tramer, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: USENIX Security Symposium, pp. 601–618. USENIX Association (2016)
31. Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhancing Technol. **2019**(3), 26–49 (2019)
32. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. In: IEEE Symposium on Security and Privacy, pp. 36–52. IEEE Computer Society (2018)
33. Wang, W., et al.: Rafiki: machine learning as an analytics service system. Proc. VLDB Endow. **12**(2), 128–140 (2018)
34. Xu, C., et al.: Correction to: lie-x: depth image based articulated object pose estimation, tracking, and action recognition on lie groups. Int. J. Comput. Vis. **126**(8), 897 (2018)

35. Yu, X., Xu, C., Xu, L., Mei, L.: Hardening secure search in encrypted database: a kga-resistance conjunctive searchable encryption scheme from lattice. Soft. Comput. **26**(21), 11139–11151 (2022)
36. Zhang, Q., Xin, C., Wu, H.: GALA: greedy computation for linear algebra in privacy-preserved neural networks. In: NDSS. The Internet Society (2021)
37. Zhang, Q., Yang, L.T., Chen, Z.: Privacy preserving deep computation model on cloud for big data feature learning. IEEE Trans. Comput. **65**(5), 1351–1362 (2016)